//
///　　인공지능 중간고사 대체과제　　///
///　　///
///　　Iterative Lenghthening Search　　///
///　　///
///　　컴퓨터학과 2018320241 정소희　　///
//

1. 교재 Figure 3.14의 uniform cost search 알고리즘과 동일한 결과를 얻을 수 있도록, iterative deepening 원리를 적용하여 space complexity가 O(bd)인 iterative lengthening search 알고리즘을 작성하시오. (b: branching factor, d: depth)

Complete, Optimal, Space complexity = O(bd)

Uniform-Cost-Search 함수는 아래와 같은 parameter들을 사용한다.

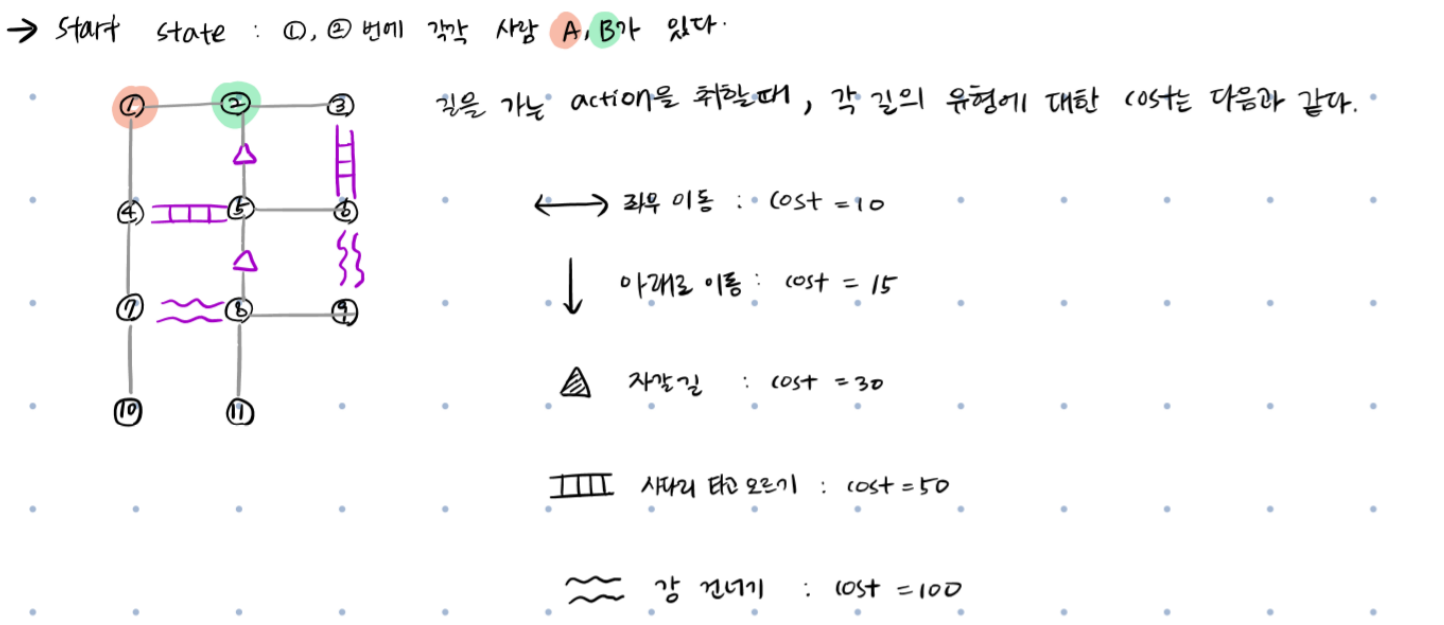
- solution_exists: ILS에서 넘겨받은 global variable이다.(bool)
- solution, min_path_cost, path_cost: ILS에서 넘겨받은 global variable이다
- limit: ILS에서 넘겨준 cost limit값이다
- state: 현재 탐색중인 state이다
- pseudocode에서는 편의상 global variable의 값을 수정할 때 주소를 접근하지 않고 변수 이름을 사용해 수정하는 것으로 표기하였다.

```
function Iterative-Lenghthening-Search(problem) returns a solution, or failure
    limit = 0
    loop
        state = initial state
        min_path_cost = ∞
        path_cost = 0            //initial path cost
        solution = ∞
        solution_exist = false

        res=Uniform-Cost-Search(state,limit,min_path_cost,path_cost,solution,solution_exist)

        if (solution_exist = true) then return solution
        else if (res != cutoff_occurred) then return failure    //cutoff not occurred & no solution
        else    limit = min_path_cost    //cutoff occurred & no solution
        //update limit: 이전 iteration에서 limit을 초과해서 discard된 path들의 cost 중
        //minimum path cost인 min_path_cost를 limit으로 한다.


function Uniform-Cost-Search (state,limit,min_path_cost,path_cost,solution,solution_exist)
                                                    returns a solution, or cutoff_occurred, or failure
    cutoff = false
    solution_exist = Goal-Test(state)    // Goal-Test(state) 함수는 현재 탐색중인 state가
```

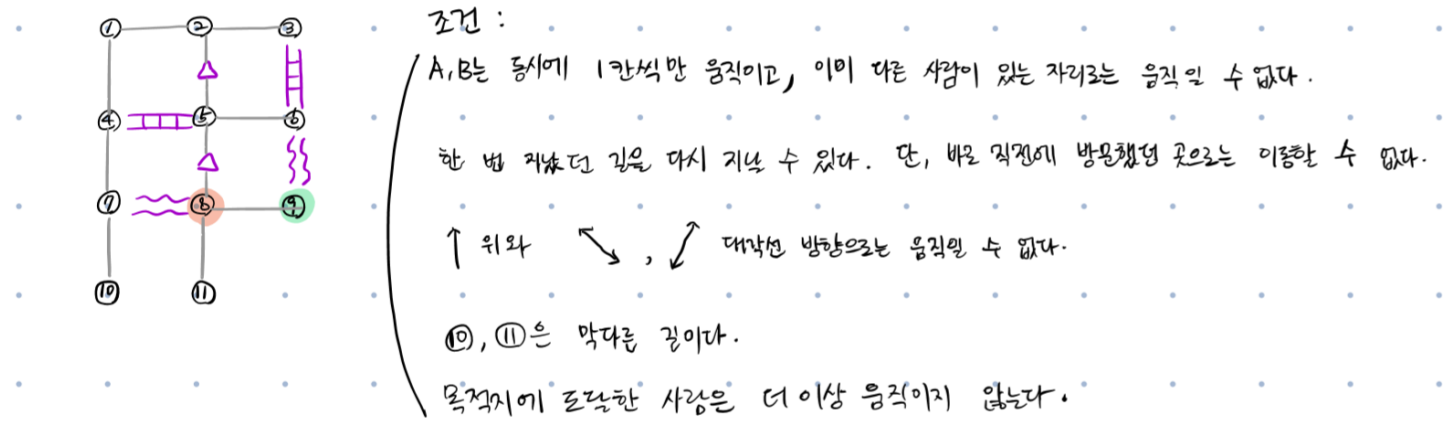
```
                                        //solution state 인지를 검사하여 true or false를 return하는 함수이다.
if(solution_exist)  //solution이 존재하면
        then   return solution = min(solution,path_cost)
        //global variable solution을 update하고 return한다. 기존의 solution과
        //새로운 solution(현재 state까지의 path_cost) 중 값이 작은 것을 solution으로 update한다.

for each actions in Actions(state)              //현재 state에서 모든 가능한 action에 대해 반복한다
        child_node = Child-Node(state,action)       //현재 state에서 action에 의해 만들어지는 child
          //Child-Node 함수는 현재 state에서 action을 취했을 때 얻어지는 child state을 return한다.
        temp_path_cost = path_cost + Calc-Cost(state,action)
          //Calc-Cost 함수는 state에서 action을 수행하는 데 필요한 cost를 return한다.
          //initial state부터 child_node까지의 total path cost가 temp_path_cost에 저장된다.

        if (temp_path_cost > limit)          //child_node까지의 path cost가 limit을 초과하면
                cutoff = true                //cutoff=true로 discard됨을 표시한다.
                min_path_cost = min(min_path_cost,temp_path_cost)
                  //min_path_cost를 이제까지 limit을 초과한 path들의
                  //path_cost 중에서 가장 작은 값으로 update한다.
                continue          //다음 child_node iteration 실행

        else      //child_node까지의 path cost가 limit보다 작거나 같으면
                res=Uniform-Cost-Search(child_node,limit,min_path_cost,
                                                        temp_path_cost,solution,solution_exist)
                if(res=cutoff_occurred) then cutoff = true
//모든 가능한 action을 검사한 후 for문이 종료된다.

if(solution_exist)  then return solution
else if(cutoff=true) then return cutoff_occurred //cutoff가 1번이상 발생함. 더 확장시켜볼 여지가 있다
else      return failure      //cutoff=false & no solution
```

2. 위에서 작성한 iterative lengthening search 알고리즘의 수행 과정을 교재 Figure 3.27과 같이 그림을 이용해서 추적하며 설명하시오. (알고리즘의 모든 경우를 그림에서 설명해야 하고, 설명에 사용한 예제는 본인이 창의적으로 만들어야 하며 교재나 인터넷 검색에서 찾을 수 없는 것이어야 함.

[문제와 문제의 규칙에 대한 설명]
> start state (initial state)의 그림과 그에 대한 설명, 그리고 cost에 대한 설명은 아래와 같다.



> goal state의 그림과 사람 A,B가 움직일 수 있는 조건에 대한 설명은 아래와 같다.



어느 한 명이라도 막다른 길에 도달하거나 나아갈 수 있는 방법이 없으면 게임에 실패한 것이다. 게임에 성공하는 경우는 위의 모든 조건을 만족하면서 사람 A,B가 각각 1->8, 2->9 번으로 움직이는 것이다.

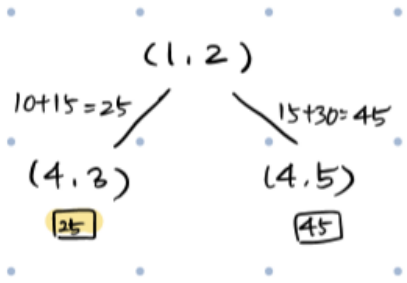
[위의 Iterative-Lengthening-Search를 사용한 수행 과정]

- 현재 state를 A,B가 위치한 곳의 번호를 나열하는 방식으로 (1,2)와 같이 표현하자.
- A,B가 path cost의 합이 최소가 되도록 state (1,2) -> state (8,9)로 이동하는 solution path를 찾는 과정을 아래에 설명하였다.
- 각 iteration에 대한 tree 형태의 그림에서 state와 state를 잇는 선분은 action을 의미하고, 각 선분 위에는 해당 action의 cost를 계산한 값을 적었다. 이는 알고리즘에서 Calc-Cost(state, action)의 return값과 같다.
- 각 iteration의 tree형태의 그림에서 각각의 node는 위의 그림과 같은 state를 나타낸다.
- leaf node의 밑에는 각각 그 state까지 도달하기 위한 path cost를 적었다. (네모로 둘러싸인 숫자 형태로 표현하였다.)
- 아래 설명에서 편의를 위해 Iterative-Lengthening-Search와 Uniform-Cost-Search 함수를 각각 ILS와 UCS로 기술하였다.

- 이제 이 문제를 수행하는 과정을 각 iteration별로 살펴보자. 여기서 각 iteration은 Iterative-Lengthening-Search에서 각각의 loop를 의미한다.

ILS iteration #1

ILS의 첫 iteration에서는 limit = 0, min_path_cost = ∞, path_cost = 0, solution_exist = false 와 initial state인 (1,2)를 Uniform-Cost-Search 함수에 parameter로 넘겨준다.



첫 iteration은 ILS에서 넘겨받은 limit = 0에서 시작한다.

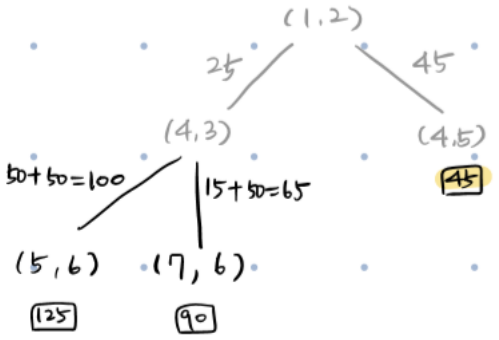
ILS에서 호출된 UCS에서 state (1,2)의 모든 가능한 action에 대해 child 하나씩 그 child까지의 path_cost가 limit을 초과하는지 검사한다. (1,2)에서 방문 가능한 child는 (4,3), (4,5) 두가지이다. (위의 그림을 참고)

이 경우, 첫 번째로 방문한 (4,3) child까지 가는 action cost는 아래로 이동 15 + 오른쪽 이동 10 = 25 이므로 (1,2)부터 (4,3)까지의 path cost는 0+25 = 25이고, 이는 limit = 0을 초과하는 값이다. limit을 초과하므로 for문 안의 if문으로 들어가고, cutoff = true로 설정되며 min_path_cost = min(∞,25) = 25로 update된다. min_path_cost를 update한 후 continue를 통해 for 문의 다음 iteration, 즉 다음 child를 검사한다.

두 번째로 방문한 (4,5) child까지 가는 action cost는 아래로 이동 15 + 자갈길 30 = 45 이므로 (1,2)부터 (4,3)까지의 path cost는 0+45 = 45dlrh, 이는 limit = 0을 초과하는 값이다. limit을 초과하므로 for문 안의 if문으로 들어가고, cutoff = true로 설정되며 min_path_cost = min(25,45) = 25로 유지된다.

모든 가능한 action에 대한 children을 검사한 후, UCS의 for문을 빠져나온다. 이 때 cutoff = true이고 solution_exist = false 이므로 cutoff_occurred를 return한다. UCS에서 cutoff_occurred가 return되면 ILS의 iteration #1에서 res = cutoff_occurred 가 되므로 ILS의 loop 안의 else 가 실행되고, limit = min_path_cost = 25로 update된다.

ILS iteration #2



두 번째 iteration은 limit = 25에서 수행된다.

iteration#1과 같이, UCS 안에서 가능한 action에 대한 모든 children을 탐색한다. 달라진 것은 limit값이 0에서 25로 증가하였다는 것이다. 이러한 변화는 for문에서 (1,2)의 child 중 (4,3)을 검사할 때 나타난다. (1,2)에서 (4,3)까지의 path cost는 25로, limit보다 크지 않다. 따라서 UCS 안의 for문 안의 else문에서 (4,3) state를 parameter로 넘겨받은 UCS 함수가 호출된다. 이 때 (1,2)부터 (4,3)까지의 path cost인 25도 UCS에 넘겨준다.

(4,3) state에 대한 UCS에서 iteration#1과 같은 작업이 수행된다. (4,3)의 children인 (5,6) state와 (7,6) state의 temp_path_cost = 25 + Calc-Cost(state,action)를 통해 (1,2)부터 (5,6), (7,6)까지의 path cost가 각각 계산된다. 계산된 path cost를 limit = 25와 비교하는데, 이 경우에는 path cost가 125와 90으로 모두 limit=25보다 작으므로 for문 안의 if문이 실행되고, if문 안에서 cutoff=true로 설정되며 min_path_cost는 ∞->125->90 순으로 update된

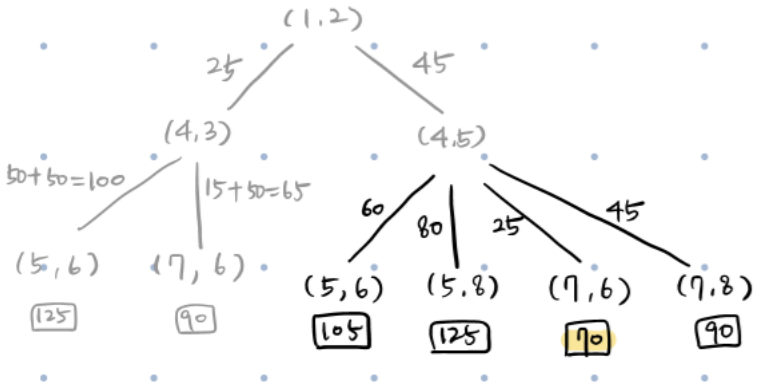
다. (4,3)의 모든 자식 state 검사가 끝나면 for문을 빠져나온다. 이 때 solution은 발견되지 않았고 cutoff=true이므로 cutoff_occurred를 return한다. (1,2)의 UCS의 else문 안에서 호출된 (4,3)의 UCS가 cutoff_occurred를 호출하므로 else문 안에서 cutoff=true가 된다.

이제 마찬가지로 (1,2)의 자식 중 (4,5)에 대한 for문 iteration이 시행된다. (4,5)까지의 path cost가 계산되고 path cost = 45 > limit = 25 이므로 if문 안에서 min_path_cost = min(90,45) = 45로 update된다.

(1,2)에 대한 모든 자식이 검사되고 for문을 빠져나온다. cutoff=true이므로 ILS로 cutoff_occurred가 return되고, limit = min_path_cost = 45로 update된다.

ILS iteration #3

세 번째 iteration은 limit = 45에서 수행된다. iteration #1과 #2에서 알고리즘의 작동 순서에 대해 충분히 설명했으므로, iteration#3 부터는 간략하게 설명하면서 숫자에 초점을 맞추려고 한다.



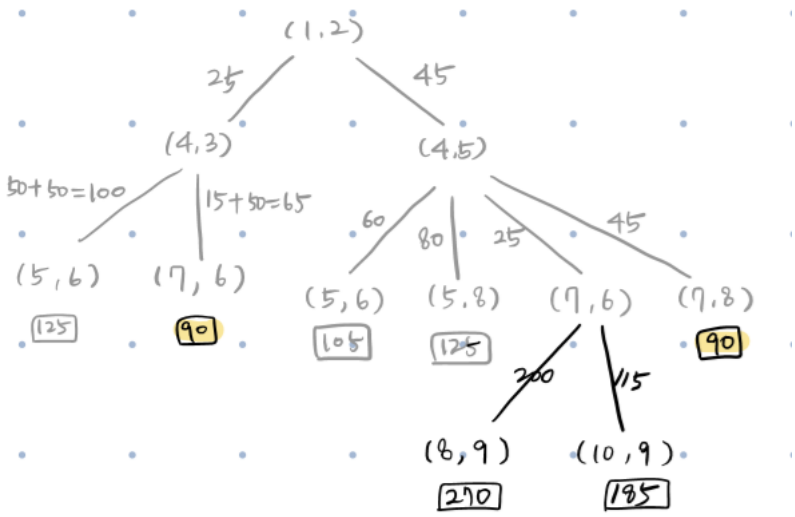
iteration#2의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 (4,5) state 뿐이다. 따라서 이미 (4,3) subtree를 모두 검사한 후 min_path_cost = 90, cutoff = true로 update가 되었다고 가정한 후, (4,5) subtree를 마저 검사해 보자.

(4,5) 까지의 path cost는 45로 limit 보다 크지 않으므로 for문의 else 안에서 (4,5)를 state로 넘겨받은 UCS가 호출된다.

(4,5)의 UCS 안에서 (4,5)의 모든 자식 (5,6), (5,8), (7,6), (7,8)이 for문을 통해 검사되는데, 각각의 (1,2)로부터의 path cost는 105, 125, 70, 90이다. min_path_cost는 70으로 update되고, cutoff = true이므로 ILS에서 limit = 70으로 update된다.

ILS iteration #4

네 번째 iteration은 limit = 70에서 수행된다. iteration#3의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 (7,6) state 뿐이다. 따라서 (7,6) state만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. (7,6) state의 자식을 확장해 보면 아래 그림과 같다.

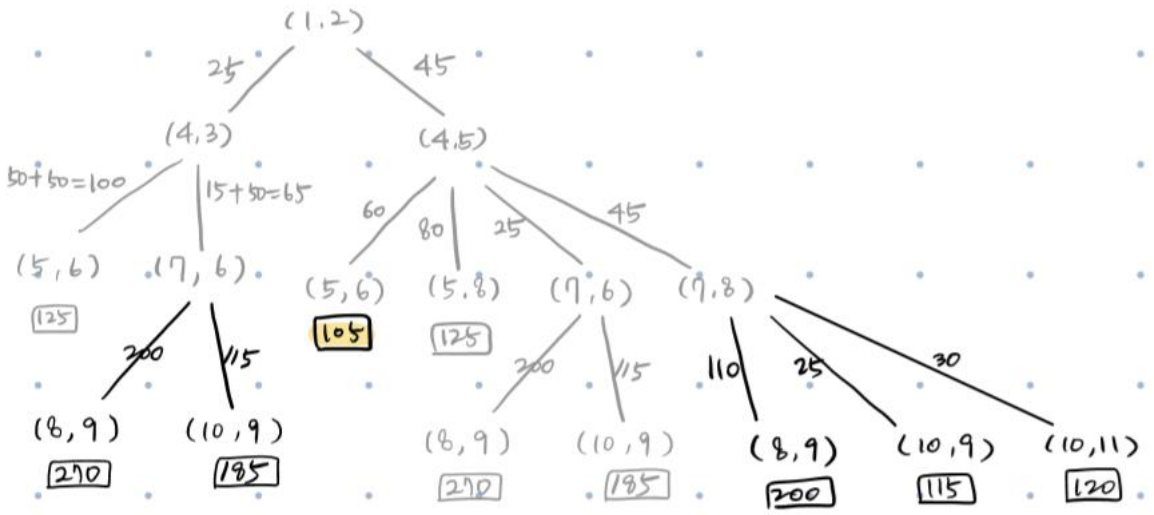


(7,6)의 두 자식 모두 (1,2)로부터의 path cost가 limit=70보다 크다.

min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것이므로 90이 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 90으로 update된다.

ILS iteration #5

다섯 번째 iteration은 limit = 90에서 수행된다. iteration#4의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 왼쪽의 (7,6) state와 (7,8) state 뿐이다. 따라서 왼쪽 (7,6) 과 (7,8) state만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. 왼쪽 (7,6) 과 (7,8) state의 자식을 확장해 보면 아래 그림과 같다.



왼쪽 (7,6)의 두 자식 모두 (1,2)로부터의 path cost가 limit=90보다 크다. (7,8)의 두 자식 모두 (1,2)로부터의 path cost가 limit=90보다 크다.

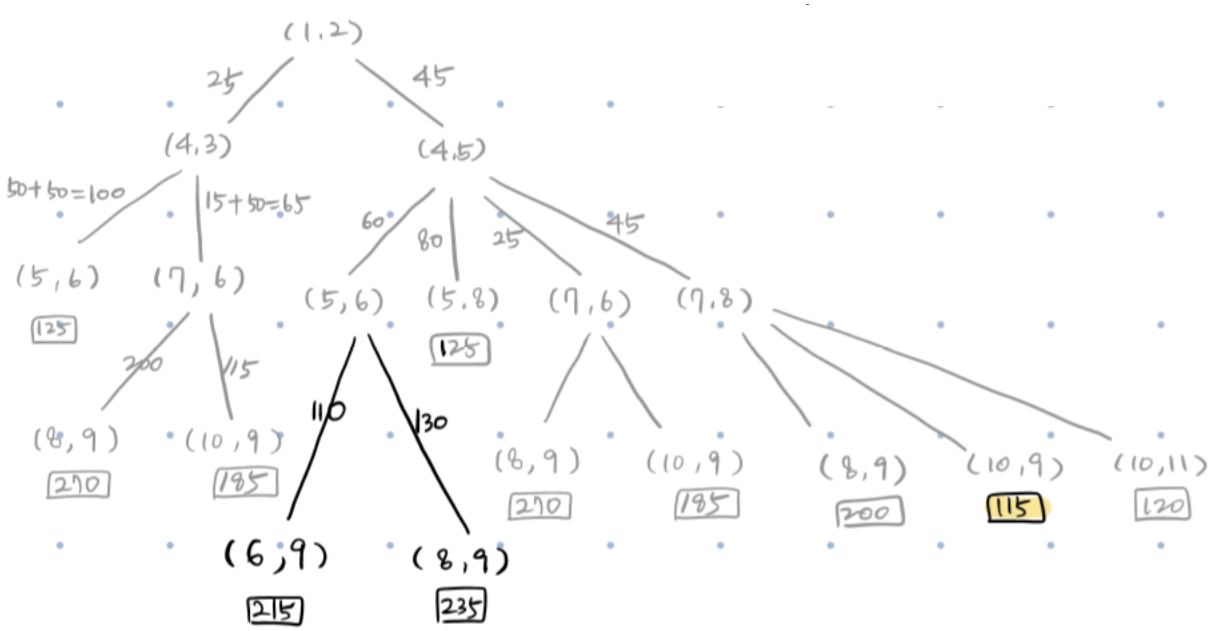
min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것이므로 105가 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 105으로 update된다.

ILS iteration #6

여섯 번째 iteration은 limit = 105에서 수행된다. iteration#5의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 오른쪽에 있는 (5,6) state 뿐이다. 따라서 오른쪽 (5,6) state만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. 오른쪽 (5,6) state의 자식을 확장해 보면 아래 그림과 같다.

오른쪽 (5,6)의 두 자식 모두 (1,2)로부터의 path cost가 limit=105보다 크다.

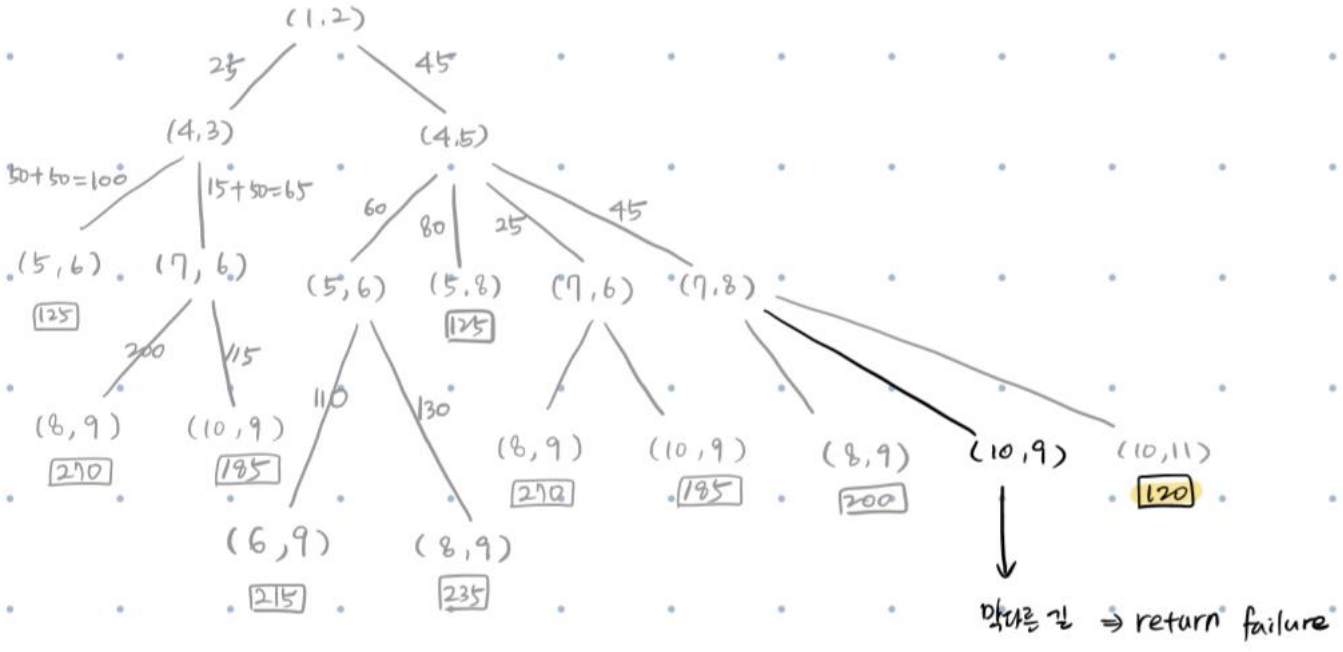
min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것이므로 115이 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 115으로 update된다.



ILS iteration #7

일곱 번째 iteration은 limit = 115에서 수행된다. iteration#6의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 제일 오른쪽에 있는 (10,9) state 뿐이다. 따라서 제일 오른쪽 (10,9) state 만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. 하지만 이 경우에는 제일 오른쪽 (10,9) state에서 더 이상 수행할 수 있는 action이 없다. 즉 제일 오른쪽 (10,9)는 막다른 길이므로 child가 없고, cutoff=false이며 solution또한 없으므로 failure가 return된다.

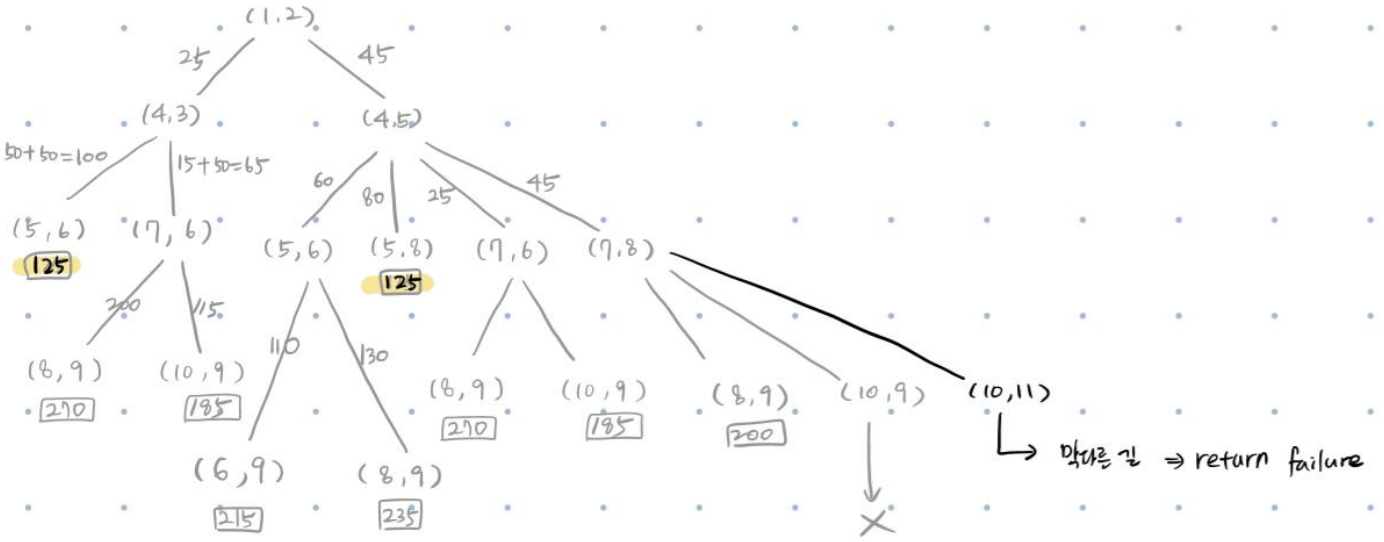
(1,2) 아래의 모든 state을 검사한 후, min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것으로 update 되므로 120이 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 120으로 update된다.



ILS iteration #8

여덟 번째 iteration은 limit = 120에서 수행된다. iteration#7의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 제일 오른쪽에 있는 (10,11) state 뿐이다. 따라서 제일 오른쪽 (10,11) state 만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. 하지만 이 경우에는 iteration#7과 동일하게 제일 오른쪽 (10,11) state에서 더 이상 수행할 수 있는 action이 없다. 즉 제일 오른쪽 (10,11)는 막다른 길이므로 child가 없고, cutoff=false이며 solution또한 없으므로 failure가 return된다.

min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것으로 update 되므로 125가 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 125로 update된다.

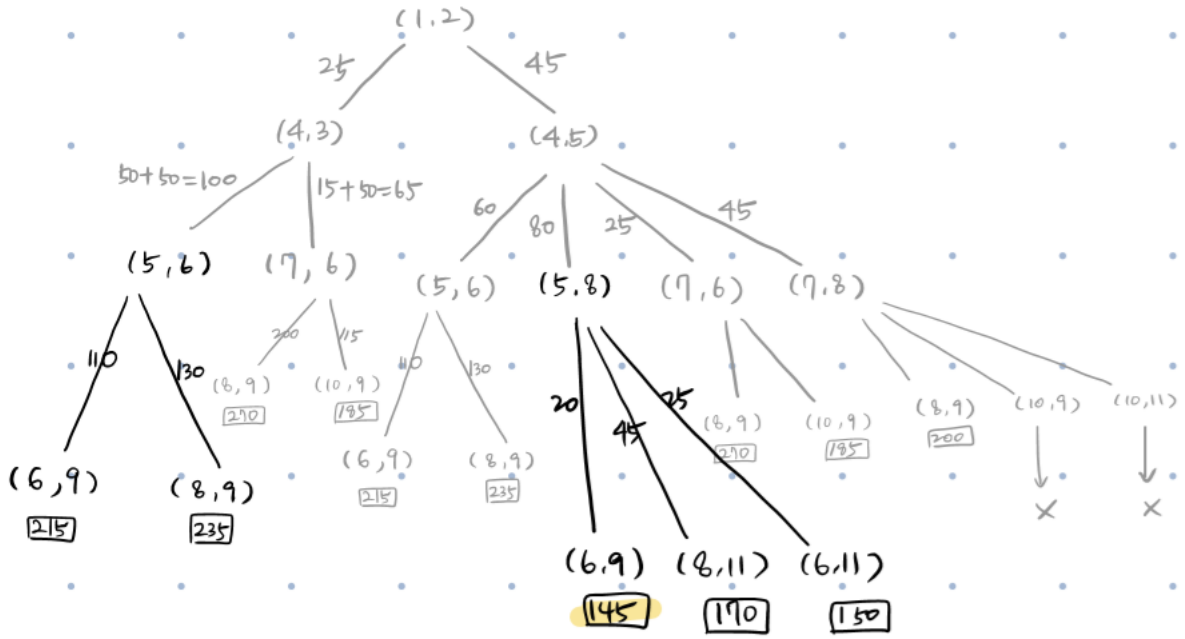


ILS iteration #9

아홉 번째 iteration은 limit = 125에서 수행된다. iteration#8의 그림에서 leaf node의 path cost중 limit보다 작거나 같은 것은 제일 왼쪽의 (5,6) state와 (5,8) state 뿐이다. 따라서 왼쪽 (5,6) state와 (5,8) state만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있고, 이것은 tree를 밑으로 더 확장한다. 왼쪽 (5,6) state와 (5,8) state의 자식을 확장해 보면 아래 그림과 같다.

왼쪽 (5,6)의 두 자식 모두 (1,2)로부터의 path cost가 limit=125보다 크다. (5,8)의 세 자식 모두 (1,2)로부터의 path cost가 limit=125보다 크다.

min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것이므로 145가 된다. cutoff = true이므로 ILS의 else에서 limit = min_path_cost = 145으로 update된다.



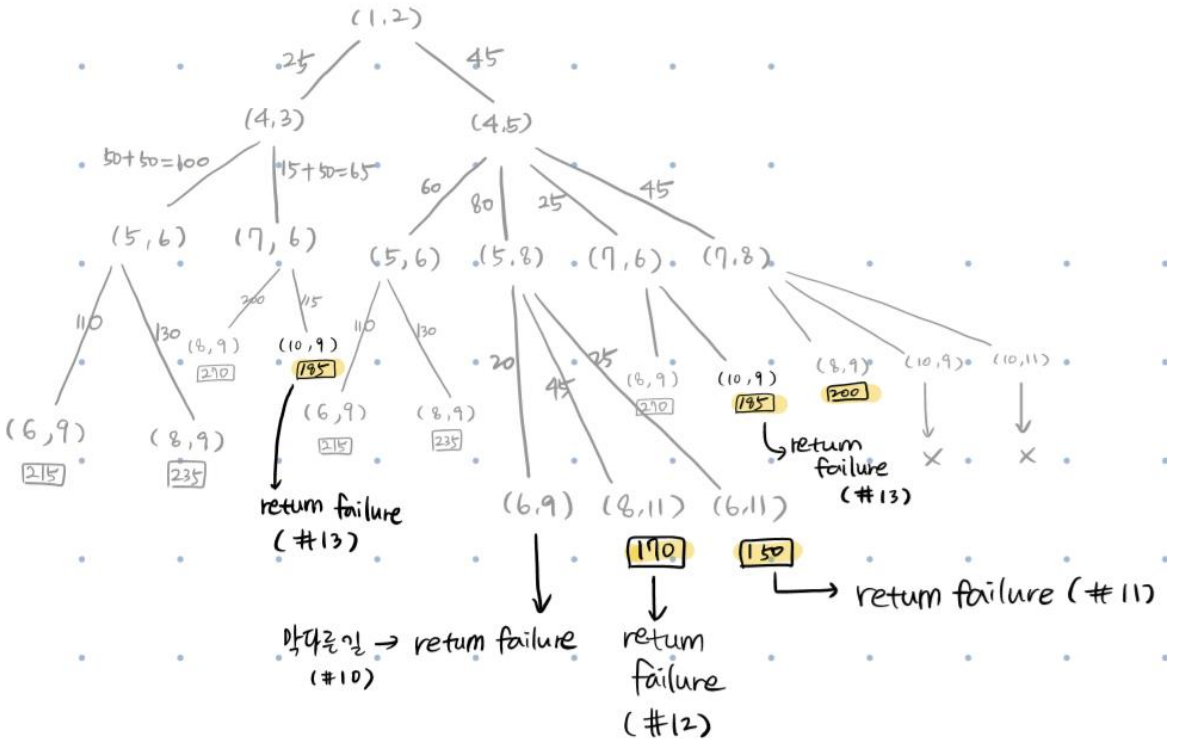
ILS iteration #10~#13

10번째 ~13번째 iteration이 수행되는 limit 조건은 아래 표와 같다.

	iteration#10	iteration#11	iteration#12	iteration#13
iteration 수행	145	150	170	185
update	150	170	185	200

iteration#10~ iteration#13은 공통적으로 leaf node까지의 path cost가 limit보다 작거나 같은 state는 모두 더 이상 수행할 수 있는 action이 없는, 막다른 길의 state이다. 따라서 해당 leaf node의 UCS에서 모두 child가 없고, cutoff=false이며 solution또한 없으므로 failure가 return된다. 하지만 다른 subtree에서 cutoff=true인 state가 많으므로 leaf node 하나에서 return된 failure는 전체 결과에 영향을 미치지 않는다.

각 iteration에서 min_path_cost는 네모로 둘러싸인 숫자들 중 가장 작은 것으로 update 되므로 각각 150, 170, 185, 200이 된다. 각 iteration은 모두 cutoff = true이므로 ILS의 else에서 각각 limit = min_path_cost = 150, 170, 185, 200으로 update된다.



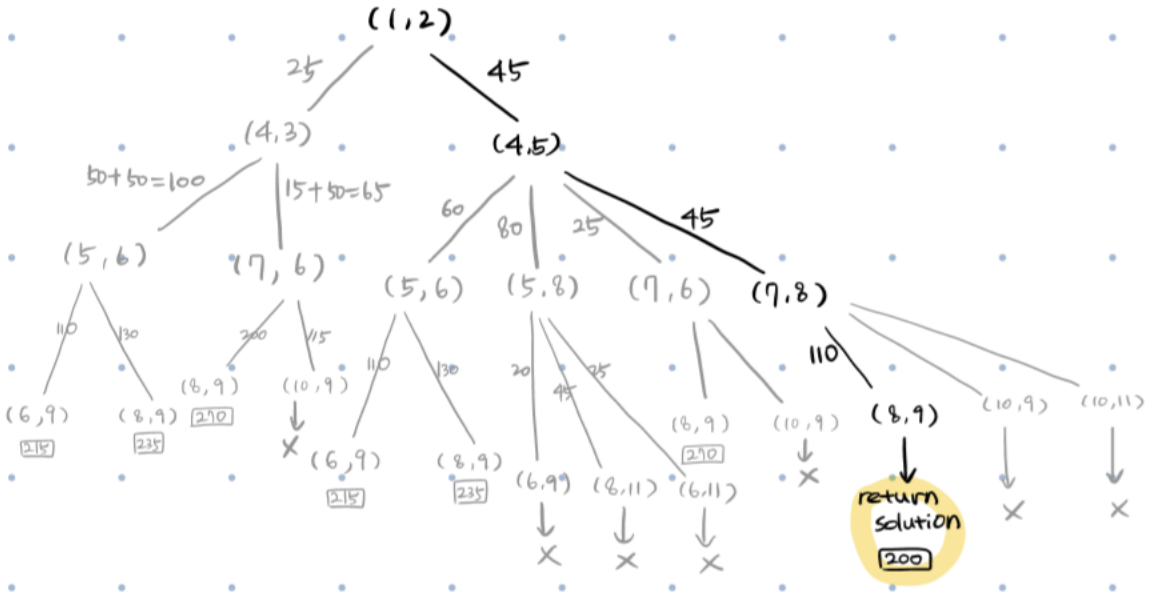
ILS iteration #14

열 네 번째 iteration은 limit = 200에서 수행된다. iteration#13의 그림에서 leaf node의 path cost중 limit=200보다 작거나 같은 것은 제일 오른쪽의 (8,9) state 뿐이다. 따라서 오른쪽 (8,9) state만 for 문의 else로 들어가서 UCS를 추가로 호출할 수 있다.

제일 오른쪽 (7,8)의 for문 안의 else문 안에서 (8,9) state의 UCS가 호출된다. UCS에서 Goal-Test(state)가 이루어져서 해당 state가 solution state인지 검사한다. 이 때 (8,9)는 goal state이므로 solution_exist = true가 되고 *if(solution_exist) then solution = min(solution,path_cost)*에 의해 solution = min(∞,200) = 200으로 update된 후 return된다.

solution_exist는 global variable로 (8,9) UCS부터 (1,2) UCS로 return될 때까지 모두 true가 유지된다. 그러므로 모든 solution path 내의 UCS에서는 *if(solution_exist) then return solution*에 의해 solution이 return된다.

결국 ILS까지 return된 solution은 ILS의 14번째 loop에서 *if(solution_exist = true) then return solution*에 의해 최종적으로 아래 그림과 같은 최고 cost를 갖는 solution path가 ILS에서 return된다.


(1,2)
25
45
(4,3)
(4,5)
50+50=100
15+50=65
60
80
25
45
(5,6)
(7,6)
(5,6)
(5,8)
(7,6)
(7,8)
110
130
200
115
110
130
20
45
25
110
(6,9)
215
(8,9)
235
(8,9)
270
(10,9)
(6,9)
215
(8,9)
235
(6,9)
(8,11)
(6,11)
(8,9)
270
(10,9)
(8,9)
(10,9)
(10,11)
return solution
200